SUP'COM
INDP2
Traitements multimédia intelligents
Sofia BEN JEBARA
2021/2022

Détection de langue de rédaction d'un texte en utilisant le concept de traitement naturel du langage

1. Objectifs

Ce travail vise le développement d'une solution logicielle permettant la détection de la langue d'un texte écrit. Elle fait appel à des concepts du traitement du langage naturel (natural langage processing (NLP) et de l'apprentissage machine (machine learning ML).

2. Jeu de données

On utilisera un ensemble de données de texte écrits en 17 langues différentes, c'est-à-dire qu'on créera un modèle NLP pour prédire 17 langues différentes. Il s'agit des langues suivantes : Allemand, Anglais, Arabe, Danois, Espagnol, Français, Grec, Hindi (la plus parlée en Inde), Italien, Kannada (état de Karnataka en Inde), Malayalam (sud de l'Inde), Néerlandais, Portugais, Russe, Suédois, Tamoul (parlé à Sri Lanka, Malaisie, Singapour, Inde) et Turc.

Le fichier 'language_detection.csv' contient le même jeu de données composé de 10 267 données organisées en deux colonnes, la première appelée 'text' contient un texte écrit tandis que la seconde appelée 'language' est l'information de la langue utilisée pour rédiger le texte.

Ouvrir le fichier fourni avec excel et balayer son contenu tout en visualisant les textes en différentes langues et en explorant les symboles de langues particulières.

Avant de commencer la lecture des données sous Python, on importe les bibliothèques nécessaires, en l'occurrence pandas :

```
import pandas as pd
```

Ensuite, on lit le jeu de données comme suit (tout en pointant sur le répertoire où se trouve le fichier):

```
data = pd.read_csv("Reperoire/language_detection.csv")
```

a. Observer le contenu des 10 premières lignes du jeu de données :

```
data.head(10)
```

On peut compter le nombre de données de chaque langue comme suit :

```
data["Language"].value_counts()
```

b. Traduire en pourcentage.

3. Séparation des variables indépendantes et dépendantes

Maintenant, nous pouvons séparer les variables dépendantes et indépendantes. Ici, les données de texte constituent la variable indépendante tandis que le nom de la langue est la variable dépendante. Sur le dataframe utilisé, l'opération se fait comme suit :

```
X = data["Text"]
y = data["Language"]
```

4. Encodage de la variable 'Langue'

La variable de sortie (variable dépendante), qui est le nom de la langue, est une variable catégorielle. Pour entraîner le modèle de détection de la langue, elle doit être convertie sous une forme numérique. Nous effectuons ainsi un encodage d'étiquette. Pour ce processus, nous importons LabelEncoder depuis sklearn et utilisons les instructions LabelEncoder et fit_transform(y) comme suit :

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y = le.fit_transform(y)
```

5. Prétraitement de texte

L'ensemble de données utilisé a été obtenu par 'Web scrapping' (grattage du web) qui est un processus spécifique d'extraction de données à partir du web. Dans ce cas d'étude, le texte a été extrait de pages Wikipédia. Il contient de nombreux symboles indésirables et des nombres qui peuvent affecter la qualité du modèle. Pour cela, nous utilisons la fonction re.sub de la bibliothèque re qui permet de remplacer les occurrences d'une sous-chaîne de caractètres particulière par une autre sous-chaîne. Cette fonction prend en entrée les paramètres suivants : la sous-chaîne à remplacer, la sous-chaîne de remplacement et le texte original. Ci-dessous, un exemple de syntaxe est présenté :

```
text = re.sub(r'[!@#$(),n"%^*?:;~`0-9]', ' ', text)
```

Cette instruction permet de substituer tous les symboles disponibles dans le premier paramètre de la fonction qui sont situés à l'intérieur de r'[ ]' (à savoir ! @ # $ ( ) , n " % ^ * ? : ; etc) par un espace (paramètre du milieu entre ' ') dans la variable text (dernier paramètre).

En ce qui concerne notre cas d'études, nous allons utiliser ce bout de code qui permet d'itérer tout au long du jeu de données l'ensemble des pré-traitements de substitution, tout en rallongeant la liste de le nouvelle donnée pre-traitée à chaque itération (grâce à l'instruction data_list.append(text)):

```
import re
data_list = []
for text in X:
    text = re.sub(r'[!@#$(),n"%^*?:;~`0-9]', ' ', text)
    text = re.sub(r'[[]]', ' ', text)
    text = text.lower()
    data_list.append(text)
```

c. A la lecture du bout de code précédent, quels sont les autres pré-traitements appliqués ?
d. Choisir quelques exemples de texte qui ont besoin des pré-traitements précédents. Les visualiser avant et après traitements.

6. Sac de mots (Bag of words)

La variable de sortie du nom de la langue a été convertie en format numérique. Il devrait être de même pour la donnée qui a le format de texte. Pour cela, nous convertissons le texte sous forme numérique en créant un modèle de sac de mots.

Le concept du sac de mots est une représentation simplificatrice très connue utilisée dans le traitement du langage naturel qui permet de traduire un document sous forme d'un ensemble de mots qui appartiennent au dictionnaire du langage naturel considéré. Le modèle de sac de mots peut être utilisé dans les méthodes de détection de langue des documents pour lesquelles la fréquence d'occurrence de chaque mot est utilisée comme descripteur pour la formation d'un classifieur, sans tenir compte ni de la grammaire ni de l'ordre des mots.

Pour illustrer ce concept, considérons l'exemple de représentation numérique d'un sac de mots formé de 4 mots qui sont les noms des animaux suivants : chien, chat, poisson et oiseau. Considéos aussi le fait que nous disposons de 3 textes : "chien chat poisson ", "chien chat chat " et "poisson oiseau oiseau". La représentation numérique sous forme de vecteur de comptage de mots des 3 texte s'écrits est la suivante: [1 1 0 1], [2 1 0 0] et [0 0 2 1]. En effet, chaque vecteur est de taille 4. Le 1er chiffre est la fréquence d'occurrence du mot chien (1er mot du dictionnaire), etc.

Cet exemple est traduit par code Python suivant:

```
from sklearn.feature_extraction.text import CountVectorizer
texts=["chien chat poisson ", "chien chat chat " , "poisson oiseau oiseau"]
cv = CountVectorizer()
cv_fit=cv.fit_transform(texts)
print(cv.get_feature_names())
print(cv_fit.toarray())
```

e. Tester ce code dans un fichier séparé et observer le résultat.

Pour l'exercice de détection de langue, la numérisation du texte est faite comme suit :

```
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer()
X = cv.fit_transform(data_list).toarray()
X.shape
```

data_list n'est plus utile, on peut la supprimer pour libérer l'espace mémoire :

```
import gc
del data_list
gc.collect()
```

f. Quelle est la taille du vecteur du comptage du nombre de mots ? Comment l'argumenter ? est-il possible de visualiser les vecteurs des sacs de mots ? Pourquoi ?

7. Fractionnement de la base de données en données d'apprentissage et en données de test

Les variables d'entrée et de sortie ont été converties en données numériques. L'étape suivante consiste à créer l'ensemble d'apprentissage, pour créer le modèle de classification et l'ensemble de test, pour évaluer les performances du classifieur conçu. Avant de le faire et à cause de la taille des données relativement importante qui ne peut pas être supportée par certains Pc portables, il est recommandé de modifier le nombre de bits alloués pour chaque variable, tout en respectant les formats autorisés par Python. La variable y prend 17 valeurs possibles, on peut se contenter de la stocker sur 8 bits. La matrice X peut prendre le format d'entier 16 bits (2^{16}=65536<taille des données).

Pour ce processus, nous manipulons les données comme suit :

```
from sklearn.model_selection import train_test_split
import numpy as np
y = y.astype(np.int8)
X = X.astype(np.int16)
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size = 0.20)
```

g. A quoi correspond le nombre 0.2 dans la ligne de code précédente ?
h. Quelle est la taille de la séquence de test et celle d'apprentissage ?

8. Entraînement du modèle et prédiction (classification)

Pour la création du modèle, nous utilisons l'algorithme naive_bayes avec les données d'apprentissage :

```
from sklearn.naive_bayes import MultinomialNB
model = MultinomialNB()
model.fit(x_train, y_train)
```

Ce modèle peut être sauvegardé sur le disque comme suit :

```
import pickle
filename = 'language_detection_model.sav'
pickle.dump(model, open(filename, 'wb'))
```

Ceci a l'avantage d'éviter de chercher à le créer lorsqu'on veut ré-utiliser pour tester de nouvelles données.

Remaque : à cause de la taille des données relativement importante et la complexité du modèle bayésien à créer, certains machines informatiques ne disposent pas des ressources matérielles

adéquates (RAM par exemple). Pour cela, le modèle a été entrainé pour vous. Il suffit de l'importer :

```
filename = 'language_detection_model.sav'
model = pickle.load(open(filename, 'rb'))
```

La prédiction (classification) des données de test se fait comme suit :

```
y_pred = model.predict(x_test)
```

9. Evaluation des performances

On évalue le modèle en utilisant la métrique pércision (accuracy) et la matrice de confusion exprimées en pourcentage :

```
from sklearn.metrics import accuracy_score, confusion_matrix
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

ac = 100*accuracy_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)
cm=100*cm/ cm.astype(np.float).sum(axis=1)

plt.figure(figsize=(15,10))
sns.heatmap(cm, annot = True)
plt.show()
```

i. Comment vous semblent les prformances globales du système de détection des langues ?
j. Quelles sont les langues parfaitement classées ?
k. Quelles sont les langues pour lesquelles il y'a des erreurs de classification ? Avec quelles langues sont elles confondues ?

10. Prédiction avec de nouvelles données

Testons maintenant la prédiction du modèle en utilisant du texte dans différentes langues qui ne faisait pas partie du jeu de données initiales. Pour cela, on écrit d'abord une fonction de prédiction :

```
def predict(text):
    x = cv.transform([text]).toarray()
    lang = model.predict(x)
    lang = le.inverse_transform(lang)
    print("The langauge is in",lang[0])
```

On appelle cette fonction sur les textes suivants :

predict("Est-ce que cet exercice vous a permis d'avoir un aperçu introductif au traitement naturel du langage ?")

predict("Did this exercise give you an introductory overview to natural language processing?")

predict("Bu alıştırma size doğal dil işlemeye giriş niteliğinde bir genel bakış sağladı mı?")

predict("هل أعطاك هذا التمرين نظرة عامة تمهيدية حول معالجة اللغة الطبيعية؟")

predict("¿Este ejercicio le brindó una introducción al procesamiento del lenguaje natural?")

predict("ഈ വ്യായാമം നിങ്ങൾക്ക് സ്വാഭാവിക ഭാഷാ പ്രോസസ്സിംഗിന്റെ ഒരു ആമുഖ അവലോകനം നൽകിയോ?")

predict("Это упражнение дало вам вводный обзор обработки естественного языка?")

l. Vérifier l’exactitude de la prédiction obtenue sur le web.